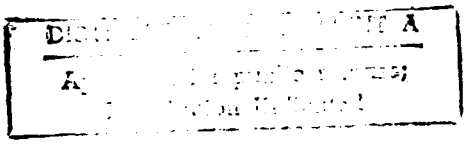
DAITC

DOD GATEWAY INFORMATION SYSTEM (DGIS) COMMON COMMAND LANGUAGE: A RETROSPECTIVE ON THE INTRODUCTION OF PROLOG AS THE DEVELOPMENT TOOL

DAITC/TR-89/6

Duc T. Tran
Allan D. Kuhn
Randy L. Bixby



May 1989

Defense Applied Information Technology Center

1800 North Beauregard Street
ndria, Virginia 22311
(703) 998-4787
No. (703) 931-3968

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DAITC/TR-89/6	5. MONITORING ORGANIZATION REPORT NUMBER(S) DTIC/TR-89/16
6a. NAME OF PERFORMING ORGANIZATION Defense Applied Information Technology Center 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Defense Technical Information Center
6c. ADDRESS (City, State, and ZIP Code) 1800 N. Beauregard Street Alexandria, VA 22311-1784	7b. ADDRESS (City, State, and ZIP Code) Cameron Station Alexandria, VA 22304-6145
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DoD Gateway Information System (DGIS) Common Command Language: A Retrospective on the Introduction of PROLOG as the Development Tool

12. PERSONAL AUTHOR(S)
Duc T. Tran, Allan D. Kuhn, Randy L. Bixby

13a. TYPE OF REPORT	13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
		8905	20

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
5	2	
12	5	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
PROLOG, Artificial Intelligence, Common Command Language, CCL, Networks, Gateways, Interfaces, DoD Gateway Information System, DGIS

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The introduction of PROLOG into the Common Command Language (CCL) development is reviewed. The advantages of using PROLOG as an artificial intelligence tool for developing fifth-generation programs are shown, in comparison with doing traditional third-generation programming. A brief is given on actual PROLOG programming, then the reasons for going to PROLOG are explained. The approach to CCL as using knowledge bases and blackboard architecture is described. The goals of CCL as an interface on the DoD Gateway Information System (DGIS) are to provide a standard program for accessing databases, provide maintainability with syntax and semantic changes as they occur, and provide adaptability to future requirements and enhancements.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Allan D. Kuhn	22b. TELEPHONE (Include Area Code) (703) 998-4600 22c. OFFICE SYMBOL DAITC Hyp Lab

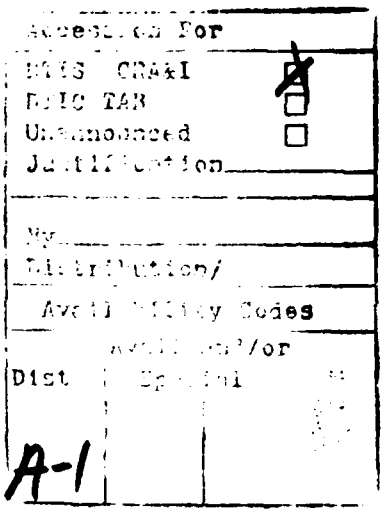
DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

DOD GATEWAY INFORMATION SYSTEM (DGIS)
COMMON COMMAND LANGUAGE:
A RETROSPECTIVE ON THE INTRODUCTION OF PROLOG AS THE DEVELOPMENT TOOL

Duc T. Tran
Allan D. Kuhn
Randy L. Bixby

May 1989



DAITC Hypermedia Laboratory Report Number 4
DTIC CCL Report No. 4

The Hypermedia Laboratory
Defense Applied Information Technology Center
Alexandria, VA 22312

Defense Technical Information Center
Alexandria, VA 22304

DOD GATEWAY INFORMATION SYSTEM (DGIS) COMMON COMMAND LANGUAGE: A RETROSPECTIVE ON THE INTRODUCTION OF PROLOG AS THE DEVELOPMENT TOOL

Duc. T. Tran, Allan D. Kuhn, and Randy L. Bixby

May 1989

Keywords: PROLOG, Artificial Intelligence, Common Command Language, CCL, Networks, Gateways, Interfaces, DoD Gateway Information System, DGIS

Abstract: The introduction of PROLOG into the Common Command Language (CCL) development is reviewed. The advantages of using PROLOG as an artificial intelligence tool for developing fifth-generation programs are shown, in comparison with doing traditional third generation programming. A brief is given on actual PROLOG programming, then the reasons for going to PROLOG are explained. The approach to CCL as using knowledge bases and blackboard architecture is described. The goals of CCL as an interface on the DoD Gateway Information System (DGIS) are to provide a standard program for accessing databases, provide maintainability with syntax and semantic changes as they occur, and provide adaptability to future requirements and enhancements.

DOD GATEWAY INFORMATION SYSTEM (DGIS) COMMON COMMAND LANGUAGE: A RETROSPECTIVE ON THE INTRODUCTION OF PROLOG AS THE DEVELOPMENT TOOL

Duc T. Tran, Allan D. Kuhn, and Randy L. Bixby

Hypermedia Laboratory
Defense Applied Information Technology Center (DAITC)

1. BACKGROUND

The Defense Technical Information Center (DTIC) has sponsored gateway technology development for the Department of Defense (DoD) since 1982. The machine for the DoD Gateway Information System (DGIS) was installed in January 1986. The purpose of DGIS is to provide users in the Department of Defense a simple, efficient, and yet powerful means to access information residing in governmental and commercial databases. Within the system the user not only has help in accessing the information but also can manipulate the results afterward. The DGIS Common Command Language (CCL) is a project generated from the realized need within the DGIS user community to access the multiplicity of information systems with a standard command language. DTIC therefore began the CCL project in 1986.

The DGIS Common Command Language (CCL) is a database manipulation language for querying databases in remote and diverse information systems. The DGIS CCL program accepts user-input in CCL and translates it into the command language of the target database. The DGIS CCL development was done to support the mission of DGIS, which is to give users an easy-to-use means of accessing the multiplicity of databases and collect information from them.

The DGIS is a UNIX-based system. Initially, the CCL began with UNIX/C programming. There are two language development tools for developing translators in the UNIX environment:

LEX -- a lexical analyzer that accepts lexical rules specified in BNF forms to determine tokens.

YACC -- a compiler-compiler; it consists of the syntax rules and the actions to be taken in recognizing a sequence of input tokens.

The input of LEX is the language to be translated. The output of LEX is a sequence of tokens that will be the input of YACC. Based on the tokens of LEX, YACC produces the translated language or error messages.

Together these two programs are standard tools for translation. They are also table-driven, that is, lexical and syntactical rules are specified in BNF grammar and thus, they should be easy to adapt and port to for different but similar translation jobs. The "different but similar" describes the circumstances of CCL.

A critical problem for a common command language interface, however, is establishing CCL command sequence memory in accessing external databases with the interface. Without this capability a command continuity sequence cannot take place in querying the remote database.

The DGIS would consider every command invoked by the DGIS CCL interface a new and original command. Consequently, each command would be a unique function entity to the database.

A simple example for the need to have a command continuity sequence is in using the 'COMBINE' results function of the CCL interface. Without having a command continuity sequence in the interface, there are never any previous results to combine.

By no means not disregarding the usefulness of LEX and YACC in third generation level C programming, in this paper we present the advantages of **PROLOG** for fifth generation programming in resolving the needs of CCL. The next section of this paper is a brief introduction to PROLOG for those who are not familiar with it.

2. INTRODUCTION TO PROLOG

PROLOG is a programming language based on logic. PROLOG programming language is based on first order logic. First order logic means that predicates can occur only as predicates, a term referring to fact assertion (see further). In logic programming language each statement is expressed as a first-order-logic axiom and the computation is a constructive proof of a goal statement from the program. PROLOG software packages are implementations of PROLOG logic programming language. Such a package should have a complete development environment that includes a debugger, an interpreter, a compiler, and an editor.

In this section we introduce PROLOG and its employment of logic programming by showing examples*. Logic concerns the methods and principles used to distinguish correct from incorrect reasoning. First consider, therefore, the following classic Greek example of logical thinking:

Every human is mortal.
Socrates is a human.
Therefore, Socrates is mortal.

This can be rephrased in logical terms as

For every X, X is mortal if X is a human.
Socrates is a human.
Therefore, Socrates is mortal.

In PROLOG the first two statements are rewritten as

```
mortal(X) :- human(X).
human(socrates).
```

In these statements "X" is a variable and "socrates" is a constant. The first statement is a general rule about the deductive relationship between the predicates **mortal** and **human.** A "predicate" is something that asserts a fact about one or more entities. The second statement is a fact. A "fact" refers to a set of possible states of affairs. These two statements form a complete PROLOG program that can be used to answer questions, such as

"Is Socrates mortal ?"

which translated into PROLOG is

---

* *There are several versions of PROLOG with some syntactical differences. All the examples in this paper are written in the syntax of the Edinburgh PROLOG.*

?- mortal(socrates).

to yield the answer

yes

and another question

?- mortal(X).

which reads "Is there a mortal?" (which equates to "human") that yields

X = socrates

Suppose we add one more fact into our PROLOG program above:

human(aristotle).

then we get the following session

?- mortal(X).
X = socrates
X = aristotle

Another example is the program to **append** two lists to form another list. We define a predicate of the form **append(X,Y,Z)**, where X, Y, Z are lists, and Z is the result of appending Y to X. The program can be written in PROLOG as

append([ ], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).

The first rule is for the base case where the first list is an empty list. The second rule is for the general case. These rules are the following points written in PROLOG:

[1] The first element of the first list (X) will be the first element of the third list.

[2] The tail of the first list (L1) will recursively concatenate to the tail of the second list to give the tail of the third list.

Executing the **append** program we would have

?- append([a,b,c], [d,e], X).
X = [a,b,c,d,e].

More interestingly

?- append(X, [d,e], [a,b,c,d,e]).
X = [a,b,c].

?- append([a,b,c], X, [a,b,c,d,e]).
X = [d,e].

?- append([a,b,c], [d,e], [a,b,c,d,e]).
yes

and finally

?- append(X, Y, [a,b]).
X = [ ]
Y = [a,b]

X = [a]
Y = [b]

X = [a,b]
Y = [ ]

which are the three possible sets of answers for the third argument as **[a,b]** of the **append** program.

The **append** program demonstrates several important characteristics of PROLOG:

- PROLOG supports recursion. "Recursion" is a mechanism in which a program module can call itself as a subfunction either directly or indirectly. The module eventually terminates once a non-recursive execution is made.
- PROLOG can provide more than one answer. These answers are provided one at a time when a PROLOG predicate is repeatedly executed. Traditional programming provides only one answer, that is always the same answer.
- PROLOG programs are invertible. For instance, the **append** program can be used both to append two lists if the first two terms of the lists are bound, or it could break a list into two smaller lists if only the third term is bound.

These aspects of PROLOG are discussed further in the context of CCL.

3. PROLOG AS A TOOL FOR LANGUAGE TRANSLATION

We now consider the problem of parsing a natural language sentence using PROLOG. Since the program involves testing to see if something is a sentence, let us define the predicate **sentence.** We will give it a meaning as follows:

sentence(X) means that:
X is a sequence of words forming a grammatical sentence.

We also introduce the predicates **noun_phrase,** and **verb_phrase** to express the following meanings:

noun_phrase(X) means that:
sequence X is a noun_phrase.

verb_phrase(X) means that:
sequence X is a verb_phrase.

Then we can put together a definition of **sentence** in terms of these predicates. A sequence X is a sentence if it decomposes into two subsequences Y and Z; Y is a noun_phrase and Z is a verb_phrase. Since we are representing a sequence as lists, we can use the predicate *append* of

the previous section to decompose one list into two others. We then can write†

```
sentence(X) :- append(Y,Z,X), noun_phrase(Y), verb_phrase(Z).
```

Similarly,

```
noun_phrase(X) :- append(Y,Z,X), determiner(Y), noun(Z).

verb_phrase(X) :- append(Y,Z,X), verb(Y), noun_phrase(Z).
verb_phrase(X) :- verb(X).

append([ ], L, L).
append([L1|L2], L, [L1|L3]) :- append(L2, L, L3).
```

Finally, we supply a set of facts:

```
determiner([the]).

noun([apple]).
noun([man]).

verb([eats]).
verb([sings]).
```

and the program is complete‡. It will successfully answer the question:

```
?- sentence([the,man,eats,the,apple]).
yes
```

This, in effect, shows the features in PROLOG programming for establishing CCL translation precepts, as explained next.

4. USING PROLOG AS THE CCL DEVELOPMENT TOOL

Using PROLOG as a tool to translate a CCL command into a command of a target database consists of writing a PROLOG predicate **translate(X,Y)** where X is a CCL command to be translated into Y, a command of a target database.

The **translate** predicate inherits several characteristics of a PROLOG program:

- if X is instantiated as a CCL command, then **translate** produces Y as an equivalent command of a target database, and there may be more than one such Y.

- if Y is instantiated as a command of a target database, then **translate** produces X as an equivalent CCL command, and there may be more than one such X.

---

† This parsing program demonstrates the point, and it works though rather inefficiently.

‡ For the sake of simplicity this parsing example assumes that a sentence has been broken into words represented in a list. In practice, a front end is needed to preprocess the raw sentence into words.

- if X (or Y) is partly instantiated, then **translate** produces Y (or X) as an equivalent command, and there may be more than one such Y (or X).

- if X and Y are both instantiated, then **translate** will determine if Y is the translation of X and *vice versa*.

One single PROLOG program **translate**, therefore, can provide translation in both directions -- from CCL to the command language of the target database and from the command language of the target database to CCL.

Now assume that we have two databases DB1 and DB2, and two PROLOG predicates **translateDB1** and **translateDB2** to translate a CCL command into a command of the respective databases DB1 and DB2. Then the predicate **translate** defined as

translate(X,Z) :- translateDB1(Y,X), translateDB2(Y,Z).

is a PROLOG program that translates a DB1 command to DB2 commands. With this feature the user can use his favorite command language of his familiar database (e.g., of DB1) to query the another database (e.g., DB2).

PROLOG is equipped also with the built-in predicate **consult** to read in a stored program from an external file. The read-in statements are added to the main memory database of PROLOG to affect the execution of later translations. This feature is ideal for customizing the CCL translation to a particular database or a particular user.

PROLOG is primarily an Artificial Intelligence language. One of its important applications has been the language for developing *Expert Systems.* It has been demonstrated that it is relatively a simple task to **sugar-coat** PROLOG to provide the capability to *explain* its action, provide the *history* of execution, and *alter* the behavior of the program by adding and deleting PROLOG statements. This is certainly desirable since in CCL we want to have capability in real-time to confirm the translated command, examine the translation process, and resubmit the translation with a modified translator.

5. LOOKING AT PROLOG

PROLOG -- An Artificial Intelligence Programming Tool

Per the *Catalogue of Artificial Intelligence*, Alan Bundy, Springer-Verlag 1986:

PROLOG is . . . *A simple but powerful and practical programming language based on the idea of programming in logic. PROLOG programs may be viewed as logical clauses and the interpreter as an efficient resolution theorum prover. PROLOG may be looked on as an extension of* **LISP** *in that it provides as primitives* **pattern directed procedure invocation** *and* **non-determinism** *(backtracking). It provides general recursive (tree-like) data structures that are accessed by pattern-matching rather than by explicit selector functions. There are no destructive operations on these data structures, but structures may contain empty slots (uninstantiated logical variables) which can be filled in later. There is also an* **assertional database** *which is used for relatively long-lived or permanent data.*

The Hypermedia Laboratory's entry into Artificial Intelligence (AI) came from its experiences in the development of the DGIS Common Command Language in UNIX/C on the DGIS VAX 11/780 during 1986-87. After assessing the rigidity of third generation programming (C, with UNIX utilities), coupled with the requirement for universality and a more human-like human-machine interface, we decided to make the jump to AI in April, 1987. A review of the AI tools relative to the

needs of CCL led to the decision to restructure the DGIS CCL in PROLOG, still in a mainframe environment (VAX 11/780). The reasons were as follows:

The Reversibility of PROLOG: Logic programming concerns the relationships of objects (or terms). In determining the truth of relations, we can have reversibility in programs, *that is*, one can write a program and have its inverse for free (with some restrictions). This feature provides a tremendous advantage to CCL, for example, in that CCL concerns the translation of a multiplicity of diverse search languages.

The Database Capability of PROLOG: PROLOG has its own internal database capability. This feature allows programs to manipulate codes as relations that can be asserted or deleted. The feature can also be extended to external databases (*e.g., via* a RDBMS), to achieve the flexibility of storing knowledge in both PROLOG internal databases and traditional external databases. We thus take advantage of database technology for performance and ease of use. (Commercial databases are equipped with a several utilities to maintain and access their data.)

The Separation of Logic and Control: This separation is encouraged in PROLOG. PROLOG programs can be considered an an amalgamation of rules and facts. They are governed by the default execution control of the PROLOG language, *i.e.*, backtracking by left to right and top to bottom execution. This control can be easily supplemented or replaced by more powerful meta-rules, coded also in PROLOG.

Object Inheritance & Message Passing: These are two powerful programming features in object language methodology. Object inheritance and message passing are easily implemented and embedded within PROLOG. Object language methodology has been used in CCL, in conjunction with the above capabilities of logic programming language.

> Per the Catalogue/Bundy:
>
> *Since it's adoption as a base language for the Japanese Fifth Generation Project, PROLOG has exhibited a remarkable rise in popularity. Commercial and/or research implementations are now available for almost every conceivable architecture, ranging from home micros to large scale mainframes.*

There are several PROLOG Compilers/Interpreters available for running in UNIX environments. When selecting a particular implementation for CCL, we were interested in the following features:

> PROLOG Must:
>
> - be available as compiler and interpreter,
> - interface to other high-level languages (e.g., C).
> - include in its built-in functions I/O compatibility with the UNIX environment, that is, I/O redirection, piping, etc.
>
> PROLOG Should:
>
> - follow the Edinburgh PROLOG syntax as the industry standard.
> - have good debugging facilities.
> - include user-friendly interfaces (e.g., windows).

There are two basic implementation versions in PROLOG. One is its implementation as commercial versions, the other as university versions. Following are examples, one of each.

QUINTUS PROLOG -- A Vendor Package Version

Quintus Prolog is a PROLOG commercial implementation by Quintus Computer Systems, Inc., Mountain View, California. Quintus was started by a group of researchers that wrote the world's first PROLOG compiler at University of Edinburgh, Scotland.

The Quintus implementation of PROLOG is fast and efficient and it is available on a number of UNIX computers and the DEC VMS operating system. We selected Quintus in 1987 after a period of evaluation and testing. A version of Quintus Prolog was installed on our VAX 11/780, and it has been used to implement the DGIS Common Command Language System (CCLS) and hypertext systems. In October 1988 we obtained on loan Quintus ProWINDOWS for evaluation. ProWindows is an object-oriented PROLOG-based package for Sun workstations. It could be used for implementing a bitmapped user-interface to CCLS, hypermedia, and Neural Network applications. We observed the ease of programming and expressibility provided by this product that is not generally provided elsewhere.

The major features of Quintus Prolog include:

* A fast, efficient implementation.
* Easy program entry and debugging using the Emacs text editor interface.
* Interface with programs and modules written in other languages, such as C.
* High degree of compatibility with the industry standard of PROLOG (Dec-10 PROLOG).
* On the Sun workstation Quintus provides ProWINDOWS, an object-oriented programming package that enables programmers using Quintus Prolog to quickly and easily create window-based user interfaces for their PROLOG application programs.

With Quintus Prolog on the VAX, the Hypermedia Laboratory, in addition to CCLS, has created several PROLOG-based programs. One is an implementation of hypertext processes, available as a shared computing resource (*vis-a-vis* a PC standalone package). Those programs in essence have changed the VAX to a **hyperVAX** system, and put the PROLOG-based programs into a distributed programs environment. In this application we structure the PROLOG programs as processes separate from the VAX's C frontend process. The communication protocol between PROLOG and C to send requests and retrieve answers is provided by Quintus Prolog. This facility is based on standard BSD UNIX sockets and it is useful for several reasons:

* It allows PROLOG to function as a back-end component of another system written in another language.
* It takes advantage of the computing power of a network of computers by allowing the construction of a set of cooperating PROLOG processes running on multiple machines.
* It permits PROLOG to control distributed processes.

A strong advantage in acquiring a commercial package is the inclusion of vendor maintenance and support, such as software updates, documentation updates, and vendor conferral support. This maintenance is especially critical for leading-edge technologies, such as PROLOG as an Artificial Intelligence tool, as they continue to evolve.

NU-PROLOG -- An Example of a University Version

NU-PROLOG is an example of a university PROLOG development application, *vis-a-vis* a commercial development. A university version is a research development made available mostly for non-commercial uses, whereas a commercial version is vended as a frozen version, to include options for maintenance, training, and later updates. A second major difference between university and vendor versions is that a university version usually is relatively low cost, does not include

maintenance and training, but reflects the state-of-the-art of the technology.

NU-PROLOG is a development of the Machine Intelligence Project of the University of Melbourne, Australia. The Hypermedia Laboratory's interest in this particular version stems from its capability to be installed on the Pyramid 98x, the DTIC/DAITC operational DGIS machine; contains features reflecting the latest advances in the logic programming field; and is compatible with Quintus PROLOG, the vendor package on the DAITC VAX 11/780. At the time of the transition from the VAX to the Pyramid in August, 1988, Quintus had not modified a package for the Pyramid. A review of PROLOG packages during May-June preceding brought up NU-PROLOG.

NU-PROLOG succeeds an earlier MU-PROLOG, and moves PROLOG closer to the ideals of Logic Programming by allowing the user to program in a style closer to first order logic. It provides substantial performance gains over the earlier interpreted systems such as MU-PROLOG, with the following features*:

* Compiles PROLOG programs into machine code for an enhanced version of the Warren abstract machine.
* Incorporates a database system based on superimposed codeword indexing which can store general PROLOG terms in external databases for fast retrieval; the database system makes use of the superjoin algorithm to perform efficient join operations.
* Uses "when" declarations (successor to "wait") to control execution of programs according to the availability of data.
* Implements a large set of built-in predicates, including many Quintus PROLOG predicates; most DEC-10/Edinburgh/MU-PROLOG library predicates are available through compatibility libraries.

The NU-PROLOG system contains the following major components:

* "nc", the NU-PROLOG compiler.
* "np", a simple interpreter-style interface which implements the standard Edinburgh PROLOG style debugging facilities and has a sophisticated query language for accessing external database predicates.
* "nac", a program for adding control information to programs written in a purely logical style.
* "nit", a program for reporting common errors in programs (*cf.* UNIX/C's "lint").

The NU-PROLOG 1.3 release includes an improved interpreter and debugger that understand "when" declarations, floating point arithmetic, an interface to foreign functions on many machines, and the usual large collection of bug fixes. NU-PROLOG runs under UNIX System V and Berkeley BSD UNIX 4.[23]. It has been implemented on the VAX 11/780, Pyramid 98x, Elxsi 6400, Perkin Elmer 3240, Sun workstations, and Integrated Solutions Workstations. The Hypermedia Laboratory has Suns that in turn communicate with the central VAX and Pyramid of the Center.

6. MOVING TO PROLOG

DGIS offers the user access to many information systems. Each of these systems is equipped with its own command language and procedures to access information. Often the differences between the command languages are substantial enough to pose a barrier to both casual users and professional librarians. Furthermore, the command languages are maintained

* NU-PROLOG descriptive information has been picked up verbatim from information provided by the University of Melbourne.

by a variety of vendors and government agencies wherein changes may be introduced without warning to users.

The goal of the CCL project is to solve the above problem by introducing a single command language access to bibliographic databases through the DGIS. CCL will incorporate procedures to ensure its correctness concerning changes and availability of features in a database.

The project started with the commitment to carry out the proposed CCL standard of the National Information Standards Organization (NISO). This standard evolves from the most often needed and the most popular features of existing bibliographic databases. In early 1987, several quick DGIS CCL prototypes were successfully implemented in C using the standard UNIX tools of LEX and YACC for language translation. These prototypes established the feasibility of the project and insights into the problem.

The current phase of the project uses Artificial Intelligence techniques of blackboard architecture and knowledge-based driven knowledge sources. Quintus PROLOG has been chosen to be the primary implementation language coupled with some low level system related support functions written in C. The introduction of PROLOG requires the implementation of CCL as a knowledge based system because this fits well with the anticipation of its gradual migration from a structured command language of NISO to a natural language.

7. THE APPROACH

The incorporation of PROLOG in CCL implementation has several design goals:

1) Provide a single CCL program for bibliographic data bases.

2) Provide a high degree of maintainability with the syntax and semantic changes of both CCL and the underlying command languages.

3) Provide adaptability to future enhancements being planned to the current version of CCL.

Based on these design goals, the DGIS CCL is structured as a knowledge-based system. CCL can be thought of as a black box between the user and the host database. The control program of CCL, called CP, is a blackboard-based architecture PROLOG program that controls the interaction between the CCL agents and the communication agents (we use the term "knowledge source" rather than "agent" to be consistent with the literature on blackboard systems). The CCL knowledge sources are the experts based on knowledge-based systems.

Typically there are two types of CCL knowledge bases. One is pertinent to user information, and the other is the knowledge base about databases. The user knowledge base (UKB) system stores information relevant to a particular user, or a group of users. Examples of this information are one's areas of interest (database names), short-hand (CCL scripts, aliases), ones privileges, *etc.* All this information is needed by the CCL to intelligently converse with and interpret commands from the user. The database knowledge base (DKB) contains information needed to translate CCL commands into host database commands and to understand the returning results and errors from the database.

The control program (CP) is a typical blackboard based program. It is a PROLOG implementation of an object-oriented system where the blackboard is nothing more than a general object that registers and monitors progress of the related knowledge sources. Each knowledge source is an object that is activated and deactivated by messages. The knowledge source's progress and results are also composed of messages whenever possible for the blackboard of the CP.

The DGIS environment provides an important communication facility called Network Access Machine (or NAM). CCL uses NAM to do the DGIS-to-host connections. The results from NAM

are packaged in messages to be processed by CCL.

The construction of CCL knowledge bases is the result of the cooperation with the domain experts provided by the sponsor, DTIC, to interpret the command languages and capture the librarian usage of command languages. Several PROLOG tools that help maintain and validate knowledge bases were also implemented.

8. SUSTAINING DIRECTIONS

DGIS CCL is an on-going project. Several models of the above described system have been prototyped. Work is in progress to improve the following aspects:

1) the command translation between two command languages are sometimes not one-to-one. To provide a truly uniform capability across various host databases, CCL attempts to fill in the gap of capabilities not provided by the host command language. Often this means that CCL has to maintain temporary files and results. The program also decomposes a CCL command into several primitive host commands. The execution of those host commands are structured as a planning process of actions. Each action is the execution of a single primitive host command, and the success and failure of the action can cause different plans to be considered or aborted.

2) DGIS CCL is envisioned for the gradual migration of CCL from a structured language of NISO CCL to natural language. We have prototyped a version of a bottom parser (based on the Japanese' BUP parser). It is intended to have this parser to fully cooperate with a blackboard system of the CP.

3) DGIS CCL is currently limited to a single database access. We are investigating the possibility of simultaneous database accessing. This extension will allow the user to converse with a CCL that is coupled with a directory of databases for meta-information about databases, which could then activate several CCL executions. The results would then be combined and intelligently presented to the user.

4) The design of the knowledge bases on which DGIS CCL relies has a boundaried degree of efficiency. We are investigating the knowledge base interface to a relational database system and database machine to optimize that efficiency.

The DGIS CCL now interfaces with five major information systems. Two are governmental: DROLS (Defense RDT&E On-Line System), which is maintained by the Defense Technical Information Center for the Department of Defense, and NASA/RECON, maintained by the National Aeronautical and Space Administration. Three are vendor-supplied: BRS, DIALOG, and ORBIT. These are the base systems of the project. Additional systems are expected to be added, including foreign, as DGIS gains access to them. The basic goal of CCL is not only to provide a means of searching systems in a standard, universal manner, but also to provide a means for the user to search unfamiliar systems.

9. SUMMARY

By incorporating PROLOG, The DGIS CCL has the capability to read in a program in accessing an external file. The read-in statements are added to the main memory database of PROLOG to affect the execution of later CCL command translations. This application will permit

the DGIS CCL to have the command continuity that is required to search remote databases, and do the same thing that is done in the remote databases as is done in their native command languages.

Additional advantages are:

a. PROLOG application will eventually allow reciprocal CCL-native command language translation, i.e., use of CCL or the native command language.

b. Because of the above, PROLOG application will eventually include cross-translation of native command languages. This will incorporate the capability to use the native command language of one database in another. Otherwise, this capability, originally intended in the CCL project, would take place as a later development.

c. Integrating PROLOG with UNIX on DGIS allows development of combinational features, i.e., UNIX windowing and PROLOG-applied CCL invocations.

The DGIS Common Command Language is to serve as an interface between the user and the information he seeks in the multiplicity of diverse information systems. Because of the introduction of PROLOG into the Common Command Language development, the door is now open to future artificial intelligence applications in DGIS and DTIC. The applications of artificial intelligence to information accessing and retrieval will allow DoD R&D managers, engineers, and scientists to overcome the barriers of user-system communications. They will be able to more easily access and collect the information they need. The availability of that information will make users more efficient, and help them avoid the high costs of duplicating or ignoring pertinent work. The incorporation of AI applications in DGIS will make the human-machine interface on DGIS more human-like in its functions and responses, and tolerant of human frailties that are caused by the complexities of the human mind.

The compelling rationale for going to PROLOG was to achieve the AI-based human-machine interface for the DGIS common command language. That is, an interface that simulates to a degree the processes that we think function in the human mind. Comparing our third generation C-programmed prototypes with the potential capabilities of fifth generation constructions, we saw that:

Third generation programming operates strictly in an algorithmic manner; it functions sequentially, based on *yes-no* decision points with known and predefined goals as results.

*** Fifth generation programs - artificial intelligence - function conceptually in a concurrent processing manner; there are many parallel decision points, none of them "yes"-or-"no", but rather "this" or "that".**

In algorithmic programming the programming language is the program.

*** In AI the information is the program.**

Algorithmic programming provides information.

*** AI information programming provides knowledge.**

Algorithmic programming gives you an item of information, or it doesn't.

*** AI gives you information that may or may not be pertinent toward making a decision.**

**** These processes mimic how we think the mind works.***

```
DGIS/vax[1]% ccl
------------------
Loading NASA knowledge base ...
Loading User Profile Knowledge Base ...
------------------
CCL > choose
NASA-RECON identifies its databases by number or by letter. Please consult system documentation for
proper database identification.
Illegal CCL command !!
CCL > choose a
You have chosen NASA-RECON file a
CCL > find
    select [ ]   <<<<This is the echo target database (NASA) entry
          SELECT
          TERM IN SELECT COMMAND NOT IN DICTIONARY
        ENTER:
CCL > find artificial intelligence
  select artificial intelligence   <<<<echo
        1  2718    2718 ST/ARTIFICIAL INTELLIGENCE
        ENTER:
CCL > display
Invalid CCL command.
CCL > explain display
The DISPLAY command initiates an online display of results from the immmediately preceding search
results set. Three DISPLAY commands are currently available on CCL NASA-RECON. These are: -etc.-
CCL > display short 1-3
The DISPLAY command in CCLS NASA requires specification of a set number. Please re-enter your
DISPLAY command in the format:
   DISPLAY <set no.> <format> <items>
CCL > display 1 short 1-3
 browse 1/6/1-3   <<<<echo
       BROWSE  1/6/1
    88A29425  ISSUE 11  PAGE 1738  CATEGORY 61  87/00/00  176 PAGES  In
    RUSSIAN  UNCLASSIFIED DOCUMENT     COPYRIGHT
UTTL: Pattern recognition and natural language understanding by a computer ---
    Russian book
AUTH: A/FAIN, VITALII SAMOILOVICH
    -etc.-
-------------------------
CCL > start
Current system must be terminated before another system can be started.
CCL > stop
*** Goodbye ***
[ End of Prolog execution ]

*** GOODBYE ***

DGIS/vax[2]%
```

DGIS CCLS prototype: a sample CCL search and retrieval session, including mistaken entries with CCLS responses.

DTIC | DOD GATEWAY INFORMATION SYSTEM | DGIS

COMMON COMMAND LANGUAGE SYSTEM – CCLS

Decision for AI

- Heterogeneous universe
- Third generation rigidity
- Potential for universality
- Human-machine interface

CCLS – AI

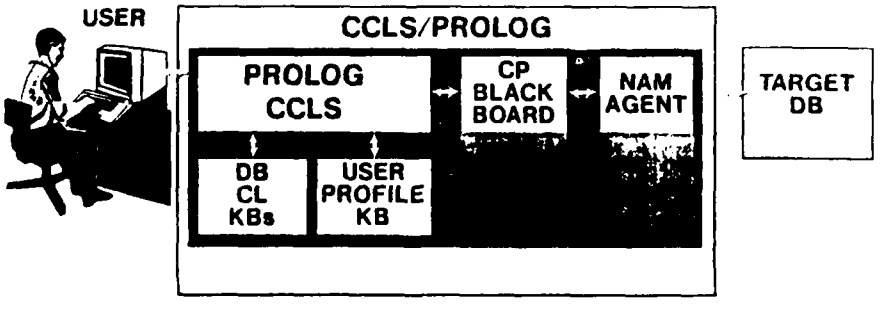
Application

- CCLS Integration with other DGIS functions
- CCLS planning capability
- Simultaneous DB access
- CCLS learning capability
- Migration to natural language

To assist the user to navigate all systems

CCLS – AI

DGIS



READING REFERENCES

The Art of PROLOG: Advanced Programming Techniques.

Leon Sterling and Ehud Shapiro. MIT Press, 1986.

Artificial Intelligence Developments RE: DoD Gateway Information System (DGIS) & Defense Applied Information Technology Center (DAITC).

Allan D. Kuhn. Defense Technical Information Center, 1987, AD-A181 101.

Catalogue of Artificial Intelligence Tools.

Alan Bundy. Second Revised Edition. Springer-Verlag, 1986.

The DoD Gateway Information System.

Gladys A. Cotter. Defense Technical Information Center, 1985, AD-A161 701.

The DoD Gateway Information System: Prototype Experience.

Gladys A. Cotter. Defense Technical Information Experience, 1986, AD-166 200.

DoD Gateway Information System (DGIS) Common Command Language: The First Prototyping and the Decision for Artificial Intelligence.

Allan D. Kuhn, Randy L. Bixby, and Duc T. Tran. Defense Technical Information Center, 1987, AD-A185 950. *Presented at and published without appendices in the Ninth National Online Meeting, New York, May 10-12 1988, pp. 169-183.*

DoD Gateway Information System (DGIS): Common Command Language Mapping.

Randy L. Bixby. Defense Technical Information Center, 1987, AD-A185 951.

DoD Gateway Information System (DGIS) Common Command Language: PROLOG Knowledge Base Profile.

Duc T. Tran. Defense Technical Information Center, 1987, AD-A150 150.

DoD Gateway Information System (DGIS) Common Command Language: The Decision for Artificial Intelligence.

Allan D. Kuhn, Randy L. Bixby, and Duc T. Tran. Hypermedia Laboratory, Defense Applied Information Technology Center, 1988, AD-A199 215. *Presented at and published in RIAO88 Conference - User-Oriented Content-Based Text and Image Handling, Massachusetts Institute of Technology, Cambridge, March 21-24 1988, pp. 863-882.*

"The DoD Gateway Information System (DGIS): The Department of Defense Microcomputer User's Gateway to the World." *Microcomputers for Information Management: An International Journal for Library and Information Services, v5 n2, June 1988, pp. 73-92.*

Allan D. Kuhn and Gladys A. Cotter. Defense Applied Information Technology Center, 1988, AD-203 351.

DoD Gateway Information System (DGIS): The Development Toward Artificial Intelligence and Hypermedia in Common Command Language.

Allan D. Kuhn. Defense Applied Information Technology Center. 1988, AD-A203 674. *Presented at and published in ONLINE INFORMATION 88: 12th International Online Information Meeting, Proceedings, London, England, 6-8 December 1988, pp. 691-704.*

Global Scientific and Technical Information Network.

Gladys A. Cotter. Defense Applied Information Technology Center, 1988, AD-A201 902. *Presented at and Published in ONLINE INFORMATION 88: 12th International Online Information Meeting, Proceedings, London, England, 6-8 December 1988, pp. 611-618.*

Intoduction to Artifical Intelligence.

Eugene Charniac, Brown University, and Drew McDermott, Yale University. Addison-Wesley Publishing Company, March 1986.

Introduction to Logic - Fifth Edition.

Irving M. Copi, University of Hawaii. MacMillan Publishing Co., Inc., 1978.

Proposed American National Standard for Information Sciences -- Common Command Language for Online Interactive Information Retrieval.

National Information Standards Organization [Z39], National Bureau of Standards. Z39.58-198x [1988].

The Scientific and Technical Information Network (STINET): Foundation for Evolution.

Gladys A. Cotter. Defense Technical Information Center, 1987, AD-A189 750. *Also presented at and published in NATO/AGARD Conference:* Barriers to Information Transfer and Approaches Toward Their Reduction, *AGARD-CCP-430; as Paper 5, "Information Retrieval Systems Evolve - Advances for Easier and More Successful Use."*

Scientific and Technical Information Network (STINET) & DoD Gateway Information System (DGIS): Reference Publications Bibliography.

Allan D. Kuhn. Defense Applied Information Technology Center, 1988, AD-A203 926.

Toward an Artificial Intelligence Environment for DTIC: Staffing Qualification Criteria for AI Application Development.

Allan D. Kuhn and Duc T. Tran. Defense Technical Information Center, 1987, AD-A181 100.

Toward an Artificial Intelligence Environment for DTIC: Proposed Tasks, Recommended Configurations, Projected Start-up Costs.

Allan D. Kuhn. Defense Technical Information Center, 1987, AD-A181 103.